Efficient Divide-and-Conquer Implementations Of Symmetric FSAs

DAVID A. G. PRITCHARD*

Department of Combinatorics & Optimization, University of Waterloo, Canada

A deterministic finite-state automaton (FSA) is an abstract sequential machine that reads the symbols comprising an input word one at a time. An FSA is *symmetric* if its output is independent of the order in which the input symbols are read, i.e., if the output is invariant under permutations of the input. We show how to convert a symmetric FSA $\mathcal{A}$ into an automaton-like divide-and-conquer process whose intermediate results are no larger than the size of $\mathcal{A}$'s memory. In comparison, a similar result for general FSA's has been long known via functional composition, but entails an exponential increase in memory size. The new result has applications to parallel processing and symmetric FSA networks.

Key words: divide and conquer, FSA, network, parallel processing, PRAM, sequential automaton, symmetry

1 INTRODUCTION

One of the simplest models of computation is the *deterministic finite state automaton* (FSA). Although FSAs are often considered to act as solitary computing devices (e.g., in the classical string matching algorithm of Knuth, Morris, and Pratt [6]) they can also be connected together to form a computing network (e.g., in cellular automata and the models of [1, 8]).

A *symmetric* automaton is one that will produce the same output even if its inputs are permuted. Symmetric FSAs are natural building blocks for

* email: daveagp@gmail.com

fault-tolerant computation networks. In previous work with Vempala [10] we showed there are symmetric FSAs implementing fault-tolerant randomized algorithms for the following tasks: implicit 2-approximate census (via an algorithm due to Milgram [8]), network search (via breadth-first search and greedy traversal), basic connectivity problems (finding all bridges), and leader election.

Precisely, the *finite-state symmetric graph automaton* (FSSGA) model introduced in [10] is that a copy of the same symmetric FSA is placed at every node of a graph (network); when a node "activates" to advance its state, it obtains one input symbol from each neighbour without regard to order. In sum, FSSGA are like symmetric cellular automata but generalized in the sense that the underlying graph does not have to be regular. Three models of symmetric automata are given in [10]: in the *sequential* model each node is a (sequential) FSA, in the *parallel* model each node uses divide-and-conquer on its inputs (in a way that will be defined precisely later), and in the *mod-thresh* model each node applies a finite-size formula (analogous to a regular expression) to update its state. One of the main results of [10] is that these three models are equivalent; e.g., for any symmetric FSA there exists a divide-and-conquer process to compute the same function. Unfortunately, for the particular construction given in [10], an exponential increase in the size of the state space is required.

More generally, an efficient way to simulate *any* FSA with divide-and-conquer has been known for decades. The basic technique is sometimes called *functional composition* as applied to *parallel prefix*. Ladner and Fischer used the technique in 1977 [7] on the PRAM model of parallel computing; see also [9] for an implementation in mesh networks. The basic idea is that for any single character σ, the transition of the FSA on that character can be viewed as function f_σ from the FSA's state space back to itself, and the computation of the FSA on a string $w = w_1 w_2 \cdots w_k$ is essentially determined by the composition of functions $f_w := f_{w_k} \circ \cdots \circ f_{w_2} \circ f_{w_1}$. In turn, this composition problem lends itself easily to divide-and-conquer: break the string into two parts $w = uv$, compute the compositions f_u and f_v for the two parts, and return $f_v \circ f_u$. Like the transformation of [10] for symmetric automata, the size of intermediate results increases exponentially, since for a state space Q there are $|Q|^{|Q|}$ functions from Q to Q.

The main contribution of this paper is that for a *symmetric* FSA, no increase in the state space size is necessary. We present the result (Theorem 7) after introducing our notation. The resulting small-state-space divide-and-conquer process is applicable to the PRAM setting, so e.g. for symmetric

FSAs we are able to decrease the working memory used by the divide-and-conquer approaches of [7, 9]. For high-degree FSSGAs and the special case of symmetric cellular automata, divide-and-conquer is a natural way for each node to read its neighbours' states, as we will illustrate in Section 2; our main result permits such divide-and-conquer processes to be more memory-efficient.

2 PRELIMINARIES

We denote an FSA by the tuple $(\Sigma, Q, q_0, \{f_\sigma\}_{\sigma\in\Sigma}, O, \beta)$. Here Σ is a finite set called the *input alphabet*, Q is a finite set called the *state space*, q_0 is an element of Q called the *initial state*, each f_σ is a function from Q to Q called the *transition function of* σ, O is a finite *output set*, and β is an *output function* from Q to O.

Definition 1 (FSA)**.** *An* FSA *is any tuple* $\mathcal{A} = (\Sigma, Q, q_0, \{f_\sigma\}_{\sigma\in\Sigma}, O, \beta)$ *of the form described above.*

Let Σ^* denote the set of all strings over Σ, and let $f \circ g$ denote the functional composition of f and g, defined by $(f \circ g)(x) = f(g(x))$. It is convenient to extend the definition of f to strings via functional composition. Namely, for a string $w = w_1 w_2 \cdots w_k$, define

$$f_w := f_{w_k} \circ f_{w_{k-1}} \circ \cdots \circ f_{w_2} \circ f_{w_1},$$

and by convention, where λ denotes the empty string, let f_λ be the identity function on Q. In particular, we obtain the identity $f_{uv}(q) = f_v(f_u(q))$ for any strings $u, v \in \Sigma^*$ and any $q \in Q$. Let Σ^+ denote the set of nonempty strings over Σ; the empty string is excluded to agree with the divide-and-conquer model later on. Our definition of f_w affords a concise description of computation for an FSA.

Definition 2 (FSA computation)**.** *An FSA* $\mathcal{A} = (\Sigma, Q, q_0, \{f_\sigma\}_{\sigma\in\Sigma}, O, \beta)$ computes *the function* $\nu_{\mathcal{A}} : \Sigma^+ \to O$ *defined by*

$$\nu_{\mathcal{A}}(w) := \beta(f_w(q_0)).$$

Note that the traditional model where the FSA accepts or rejects strings depending on the final state can be modeled by setting $O = \{accept, reject\}$ and defining $\beta(q) = accept$ iff q is an accepting state. We use the multi-output version because it is more natural in some settings, e.g., the FSSGA model.

We represent a divide-and-conquer automaton by a tuple $(\Sigma, Q, \alpha, c, O, \beta)$. As before Σ is the input alphabet, Q is the state space, O is the output set and β is the output function. Here α is an *input function* from Σ to Q and c is a *combining function* from $Q \times Q$ to Q. Informally, the divide-and-conquer automaton runs according to the following rules:

1. apply α to all input characters
2. combine states arbitrarily using c until only one state q^* is left
3. output $\beta(q^*)$.

Our definition will require that the end result of the computation is the same no matter how the arbitrary choices of combination are made.

To give our formal definition, we use a set-valued function χ that maps each nonempty string to a subset of Q so that $q^* \in \chi(w)$ iff, dividing inputs arbitrarily, the input w could produce q^* as the final state. We denote the length of w by $|w|$.

Definition 3 (DCA)**.** *Let $\mathcal{A}'$ denote the tuple $(\Sigma, Q, \alpha, c, O, \beta)$ as described above. Define $\chi_{\mathcal{A}'}(w)$ for $w \in \Sigma^+$ recursively as follows: if $|w| = 1$, say w consists of the character σ, then $\chi_{\mathcal{A}'}(w) := \{\alpha(\sigma)\}$; otherwise (for $|w| \geq 2$)*

$$\chi_{\mathcal{A}'}(w) := \bigcup_{(u,v):uv=w} \{c(q_u^*, q_v^*) \mid q_u^* \in \chi_{\mathcal{A}'}(u), q_v^* \in \chi_{\mathcal{A}'}(v)\} \qquad (1)$$

where (u, v) ranges over all partitions of w into two nonempty substrings. We say that $\mathcal{A}'$ is a divide-and-conquer automaton (DCA) *if for all $w \in \Sigma^+$,*

$$|\{\beta(q^*) \mid q^* \in \chi_{\mathcal{A}'}(w)\}| = 1. \qquad (2)$$

The previous definition amounts to saying that the output of a divide-and-conquer automaton should be well-defined regardless of how the dividing is performed. For a singleton set S let $\texttt{the\_member}(S)$ be a function that returns the element of S, i.e., it "unwraps" the set.

Definition 4 (DCA computation)**.** *A DCA $\mathcal{A}' = (\Sigma, Q, \alpha, c, O, \beta)$* computes *the function $\nu_{\mathcal{A}'} : \Sigma^+ \to O$ defined by*

$$\nu_{\mathcal{A}'}(w) = \texttt{the\_member}(\{\beta(q) \mid q \in \chi_{\mathcal{A}'}(w)\}). \qquad (3)$$

Figure 1 illustrates how a node in an FSA-based computing network could make use of the divide-and-conquer methodology. Specifically, when reading

the states of all neighbours the node can process and combine inputs from its neighbours in parallel rather than one-by-one. As a function of the neighbourhood size $|\Gamma|$ (i.e. the degree) the circuit depicted has depth $O(\log |\Gamma|)$ and hence this approach would lead to efficient physical implementation for large neighbourhoods.

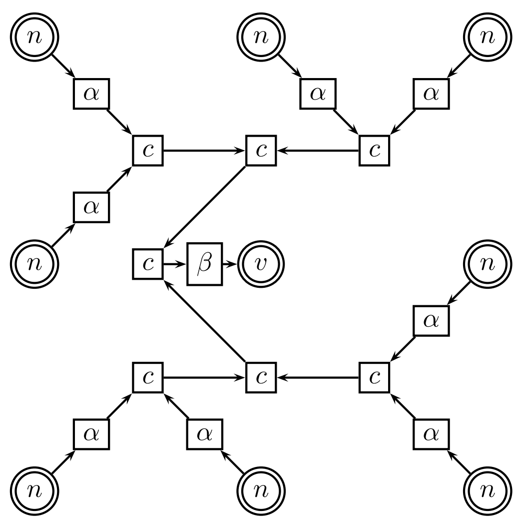

FIGURE 1
An FSA in a network updates its state via divide-and-conquer. The node v is activating and its neighbours are labeled n. The lines carry values from tail to head, and the boxes apply functions, like in a circuit diagram. Each neighbour supplies an input symbol and the divide-and-conquer process produces an output symbol which is used by v to update its state.

We denote by Q^Q the set of all functions from Q to Q. We mentioned the following well-known (e.g., [7]) result earlier:

Theorem 5. *Given any FSA $\mathcal{A}$, there is a DCA $\mathcal{A}'$ such that $\nu_{\mathcal{A}} = \nu_{\mathcal{A}'}$, i.e., $\mathcal{A}$ and $\mathcal{A}'$ compute the same function.*

Proof. Define $\mathcal{A}' = (\Sigma, Q^Q, \sigma \mapsto f_\sigma, (f_1, f_2) \mapsto f_2 \circ f_1, O, \beta)$. $\square$

Conversely, as was observed in [10], any divide-and-conquer automaton can be easily rewritten in sequential form since a sequential FSA can be thought of as conquering one input at a time.

The particular result we want to prove pertains only to symmetric automata, which we now define formally.

Definition 6. *Suppose that $\mathcal{A}$ is an FSA or a DCA. We say that $\mathcal{A}$ is* symmetric *if for every $w \in \Sigma^+$ and every permutation w' of w, $\nu_{\mathcal{A}}(w) = \nu_{\mathcal{A}}(w')$.*

The main result of the present paper is the following, which is a more efficient version of Theorem 5 for symmetric FSA's.

Theorem 7. *Given any symmetric FSA $\mathcal{A} = (\Sigma, Q, q_0, f, O, \beta)$, there is a DCA $\mathcal{A}' = (\Sigma, Q', \alpha, c, O, \beta')$ such that $\nu_{\mathcal{A}} = \nu_{\mathcal{A}'}$* and $|Q'| \leq |Q|$.

In the next section, we prove a supporting lemma for later use. In Section 4 we complete the proof of Theorem 7. In Section 5 we mention some ideas for future investigation.

3 LOOKING INSIDE A SYMMETRIC FSA

The key to Theorem 7 is to focus on automata with specific irredundany properties. Symmetry of an automaton is a *black-box property* — the definition only cares about the correspondence of final outputs when the inputs are permutations of one another, regardless of the internal structure of the automaton. We now describe how this black-box property (symmetry), when combined with irredundancy requirements, implies a structural property — namely, that the transition functions must commute.

Definition 8 ([4])**.** *Let $\mathcal{A} = (\Sigma, Q, q_0, f, O, \beta)$ be an FSA and let $q \in Q$. The state q is said to be* accessible *if for some string $w \in \Sigma^*$, $f_w(q_0) = q$. We say $\mathcal{A}$ is* accessible *if every state in Q is accessible.*

Definition 9 ([4])**.** *Let $\mathcal{A} = (\Sigma, Q, q_0, f, O, \beta)$ be an FSA and let $q, q' \in Q$. The states q and q' are said to be* distinguishable *if for some string $w \in \Sigma^*$, $\beta(f_w(q)) \neq \beta(f_w(q'))$. We say $\mathcal{A}$ is* distinguishable *if every pair of states in Q is distinguishable.*

As we will later make precise, every FSA can be rewritten in an accessible, distinguishable way. This gives some general applicability to the following lemma.

Lemma 10 (Commutativity Lemma)**.** *Let $\mathcal{A} = (\Sigma, Q, q_0, f, O, \beta)$ be a symmetric FSA that is accessible and distinguishable. Then the functions $\{f_\sigma\}_{\sigma \in \Sigma}$ commute.*

We defer the proof of the lemma to the end of this section. In order to explain how it is useful, we recall the following additional definitions.

Definition 11 ([4])**.** *Two automata $\mathcal{A}, \mathcal{A}'$ are* equivalent *if they compute the same function, i.e. if $\nu_{\mathcal{A}} = \nu_{\mathcal{A}'}$. An FSA $\mathcal{A}$ is* minimal *if for every FSA $\mathcal{A}'$ equivalent to $\mathcal{A}$, $\mathcal{A}'$ has at least as many states as $\mathcal{A}$.*

It is not hard to see that any minimal FSA must be accessible (or else we could remove some states) and distinguishable (or else we could collapse some states)*. It is also not hard to see that for every FSA $\mathcal{A}$ there exists a minimal equivalent FSA $\mathcal{A}'$; such minimization can be performed algorithmically in $poly(|Q|, |\Sigma|)$ time, e.g. using an approach of Hopcroft [5]. In sum, for any FSA we can efficiently obtain an equivalent FSA meeting the conditions of Lemma 10, which we now prove.

Proof of Lemma 10. Suppose for the sake of contradiction that not all of the functions f commute. Then $f_{\sigma_1}(f_{\sigma_2}(q)) \neq f_{\sigma_2}(f_{\sigma_1}(q))$ for some $\sigma_1, \sigma_2 \in \Sigma, q \in Q$. We want to show that this discrepancy can be "continued" to a violation of symmetry. Let q_1 denote $f_{\sigma_2}(f_{\sigma_1}(q))$ and q_2 denote $f_{\sigma_1}(f_{\sigma_2}(q))$.

First, since q is accessible, there exists some string w_ℓ such that $f_{w_\ell}(q_0) = q$. Second, since q_1 and q_2 are distinguishable, there exists some string w_r such that $\beta(f_{w_r}(q_1)) \neq \beta(f_{w_r}(q_2))$. Now putting things together we have

$$\beta(f_{w_\ell \sigma_1 \sigma_2 w_r}(q_0)) = \beta(f_{\sigma_1 \sigma_2 w_r}(q)) = \beta(f_{w_r}(q_1)).$$

Similarly

$$\beta(f_{w_\ell \sigma_2 \sigma_1 w_r}(q_0)) = \beta(f_{w_r}(q_2)) \neq \beta(f_{w_r}(q_1)).$$

Hence $\mathcal{A}$ outputs different values under the inputs $w_\ell \sigma_1 \sigma_2 w_r$ and $w_\ell \sigma_2 \sigma_1 w_r$; since these inputs are permutations of one another, this means $\mathcal{A}$ is not symmetric. $\square$

* Interestingly, the converse is also true: any accessible, distinguishable FSA is minimal. See [4] for a derivation of this result as a corollary of the Myhill-Nerode theorem; adapting the proof from accept/reject automata to our more general model is straightforward.

4 PROOF OF THEOREM 7

We are given that $\mathcal{A} = (\Sigma, Q, q_0, f, O, \beta)$ is a symmetric FSA and without loss of generality it is minimal. For each $q \in Q$, define $r[q] \in \Sigma^*$ to be a fixed *representative string* that generates state q from q_0, i.e., such that

$$f_{r[q]}(q_0) = q$$

holds. Each $r[q]$ is guaranteed to exist since q is accessible. These $r[q]$ remain fixed for the remainder of the proof.

We need the following claim, which roughly says that every string w is interchangeable with the representative string $r[f_w(q_0)]$. We know they are interchangeable when they are read first, but using the commutativity of the f's, we can show they are interchangeable when read later.

Claim 12. *For each $w \in \Sigma^*$ we have $f_w = f_{r[f_w(q_0)]}$.*

Proof. For any $q \in Q$, alternately applying the definition of $r[\cdot]$ and the commutativity of the f's, we have

$$\begin{aligned} f_w(q) &= f_w(f_{r[q]}(q_0)) = f_{r[q]}(f_w(q_0)) \\ &= f_{r[q]}(f_{r[f_w(q_0)]}(q_0)) = f_{r[f_w(q_0)]}(f_{r[q]}(q_0)) = f_{r[f_w(q_0)]}(q). \qquad \square \end{aligned}$$

4.1 The Construction

Here we define the divide-and-conquer automaton $\mathcal{A}' = (\Sigma, Q', \alpha, c, O, \beta')$. Namely, let $Q' = Q$, $\beta' = \beta$, define $\alpha(\sigma) := f_\sigma(q_0)$ and define $c(q, q') := f_{r[q']}(q)$. It remains to prove that the construction is correct, i.e., that $\nu_{\mathcal{A}} = \nu_{\mathcal{A}'}$. Our recursive proof uses the idea outlined previously, that each string w is essentially interchangeable with $r[f_w(q_0)]$.

Claim 13. *For any nonempty string $w \in \Sigma^+$, the set $\chi_{\mathcal{A}'}(w)$ is a singleton and $\texttt{the.member}(\chi_{\mathcal{A}'}(w)) = f_w(q_0)$.*

Proof. We proceed by induction on $|w|$.

Base case: If w has length 1, say it consists of the character σ, then $f_w(q_0) = f_\sigma(q_0)$, and by the definition of χ, we have $\chi_{\mathcal{A}'}(w) = \{\alpha(\sigma)\} = \{f_\sigma(q_0)\}$. Thus the claim is satisfied.

Inductive step: Now w has length 2 or more. The induction statement to be proved is $\chi_{\mathcal{A}'}(w) = \{f_w(q_0)\}$. Recalling Equation (1), which defines χ in this case, this is equivalent to saying that

$$\begin{gathered} \text{for all partitions } w = uv \text{ of } w \text{ into two nonempty substrings,} \\ c(\texttt{the.member}(\chi_{\mathcal{A}'}(u)), \texttt{the.member}(\chi_{\mathcal{A}'}(v))) = f_w(q_0). \end{gathered} \tag{4}$$

By the induction hypothesis, the left-hand side of (4) is equal to

$$c(f_u(q_0), f_v(q_0)). \tag{5}$$

Applying the definition of c, we find that the value (5) is in turn equal to $f_{r[f_v(q_0)]}(f_u(q_0))$. Finally, applying Claim 12 we see that the value (5) is equal to $f_v(f_u(q_0)) = f_w(q_0)$, as desired. □

Proof of Theorem 7. As outlined previously, minimizing $\mathcal{A}$ makes it accessible and distinguishable, without changing $\nu_{\mathcal{A}}$. Now consider the DCA $\mathcal{A}'$ as defined previously. On any input $w \in \Sigma^+$, using Claim 13,

$$\nu_{\mathcal{A}'}(w) = \beta(\texttt{the.member}(\chi_{\mathcal{A}'}(w))) = \beta(f_w(q_0)) = \nu_{\mathcal{A}}(w).$$

Hence $\mathcal{A}$ and $\mathcal{A}'$ compute the same function (i.e., they are equivalent).

Since the state space of $\mathcal{A}'$ is Q, and since Q could only have gotten smaller when $\mathcal{A}$ was minimized, the state space of the DCA $\mathcal{A}'$ is indeed no larger than the state space of the original FSA. □

One might question whether any result similar to Theorem 7 is possible if we discard the symmetry requirement. The following result gives a negative answer to this question and shows that the exponential state space increase of Theorem 5 is best possible.

Proposition 14. *For any integer $n \geq 1$, there is an n-state FSA $\mathcal{A}$ on a three-symbol alphabet Σ so that any DCA equivalent to $\mathcal{A}$ has at least n^n states.*

Proof. Let Q be a set of n states and Σ a set of size 3. Dénes [2] showed that Q^Q, viewed as a semigroup under the operation of composition, has a generating set of size 3. We choose $\{f_\sigma\}_{\sigma\in\Sigma}$ to be this generating set; this implies that for every function $g : Q \to Q$, there is a string $w[g] \in \Sigma^*$ so that $f_{w[g]} = g$. We define $O = Q$, β to be the identity function, and we choose $q_0 \in Q$ arbitrarily; this completes the definition of the FSA $\mathcal{A}$.

Suppose for the sake of contradiction that there exists a DCA $\mathcal{A}'$ that computes $\nu_{\mathcal{A}}$, and that this DCA's state space Q' has $|Q'| < n^n$. By the pigeonhole principle there are two distinct functions $g_1, g_2 \in Q^Q$ so that $\chi_{\mathcal{A}'}(w[g_1]) \cap \chi_{\mathcal{A}'}(w[g_2]) \neq \emptyset$, since each $\chi_{\mathcal{A}'}(\cdot)$ is a nonempty subset of Q'. Let $\hat{q} \in Q$ denote a state for which $g_1(\hat{q}) \neq g_2(\hat{q})$ and let $q' \in Q'$ denote any element of $\bigcap_{i=1,2} \chi_{\mathcal{A}'}(w[g_i])$.

Now let $h : Q \to Q$ be any function for which $h(q_0) = \hat{q}$. We claim that the two input strings $w[h]w[g_i]$ for $i = 1, 2$ produce different outputs under

$\mathcal{A}$ and the same output under $\mathcal{A}'$, providing the contradiction. To see that the outputs under $\mathcal{A}$ are different, observe that

$$\nu_{\mathcal{A}}(w[h]w[g_i]) = \beta(f_{w[h]w[g_i]}(q_0)) = \beta(g_i(h(q_0))) = g_i(\hat{q})$$

and since $g_1(\hat{q}) \neq g_2(\hat{q})$, we are done. To see that the outputs under $\mathcal{A}'$ are the same, let $\overline{q}'$ denote any element of $\chi_{\mathcal{A}'}(w[h])$ and notice that $c(\overline{q}', q') \in \chi_{\mathcal{A}'}(w[h]w[g_i])$ for $i = 1, 2$; then recalling Equations (2) and (3), we see that $\nu_{\mathcal{A}'}(w[h]w[g_1]) = \nu_{\mathcal{A}'}(w[h]w[g_2])$ as claimed. □

5 EXTENSIONS

We mention in this sections some extensions of FSAs and ask if analogues of Theorem 7 hold for them. Some of these issues were raised previously in [10].

First, the main result of this paper is not suitable in the following natural situation. Suppose the input alphabet and state space are both the set of all k-bit binary strings, i.e. $\Sigma = Q = \{0,1\}^k$, and that the transition function $f_\sigma(q)$ is some polynomial-time Turing-computable function of σ and q (and similarly for β). For such an FSA, $\nu_{\mathcal{A}}(w)$ can be computed in $|w| \cdot poly(k)$ time. If $\mathcal{A}$ is symmetric we can simulate it by a DCA using Theorem 7 but this approach takes exponential time in k, since minimizing $\mathcal{A}$ requires looking at all of its 2^k states. Functional composition (Theorem 5) has the same issue. Thus, the open problem is to determine if a $poly(k)$-time technique exists to convert a symmetric FSA of this type into a DCA.

Second, a variant of the above model might allow the string lengths to grow as some function $k(m)$ of the total number of inputs m. Since the original submission of this paper and independently of our work, Feldman et al. [3] showed that for this sort of model, an analogue of Theorem 7 holds where the divide-and-conquer version uses strings of length at most $k^2(m)$. Their construction, like ours, takes exponential time in $k(m)$.

Finally, the functional composition view of FSAs (e.g., in the proof of Theorem 5) also works for nondeterministic automata and probabilistic automata. A result obtained by Feldman et al. [3] shows that an analogue of Theorem 7 for probabilistic automata is false, while the nondeterministic version appears to be an open problem.

REFERENCES

[1] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. (2006). Computation in networks of passively mobile finite-state sensors. *Distributed Computing*,

18(4):235–253. Preliminary version (2004) appeared in *Proc. 23rd PODC*, pages 290–299.

[2] J. Dénes. (1968). On transformations, transformation-semigroups and graphs. In P. Erdős and G. Katona, editors, *Theory of Graphs (Proc. Colloq., Tihany, 1966)*, pages 65–75. Academic Press.

[3] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. (2008). On distributing symmetric streaming computations. In *Proc. 19th SODA*, pages 710–719.

[4] J.E. Hopcroft and J.D. Ullman. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts.

[5] John E. Hopcroft. (1971). An n log n algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The Theory of Machines and Computations*, pages 189–196. Academic Press.

[6] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. (1977). Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350.

[7] Richard E. Ladner and Michael J. Fischer. (1980). Parallel prefix computation. *J. ACM*, 27(4):831–838. Preliminary version (1977) appeared in *Proc. 6th International Conf. Parallel Processing*, pages 218–223.

[8] David L. Milgram. (1975). Web automata. *Information and Control*, 29(2):162–184.

[9] Z. George Mou and Sevan G. Ficici. (1995). A scalable divide-and-conquer parallel algorithm for finite state automata and its applications. In *Proc. 6th Conf. Parallel Processing for Scientific Computing*, pages 193–194.

[10] David Pritchard and Santosh Vempala. (2006). Symmetric network computation. In *Proc. 18th SPAA*, pages 261–270.